Introduction

This continues the programming activity for the microSD card using the Adafruit Ultimate GPS Logger Shield. This version is designed to be used for the shield revision with the 6 6-pin ICSP Header on the GPS shield and the current version of the SD library(v.1.3.0 at time of writing) from the Arduino IDE manager.

Materials List

1. Programming laptop
2. Arduino Mega
3. Assembled Adafruit GPS Shield
4. MicroSD card (32 GB or smaller)
5. USB SD card reader (if required)

NOTE: When ejecting and reinserting the SD card, be sure to close any open files. Leaving the files open will not damage anything, but it is fairly easy to look at old data and think that your code is not working. You may also wish to delete the data files in between steps to ensure you are writing new data.

Writing CSV Files

1. A more realistic use case for the SD card is writing a sequence of multiple measurements at a regular interval. For example, recording the readings from temperature and pressure sensors as the payload ascends through the tropopause and stratosphere.
2. Ideally, we want to be able to import data into data analysis software like Excel, Python, or MATLAB.
3. One common format is the comma-separated value (CSV). In CSV files, each row of data is written as a single line, and the values are divided by commas. These files can be opened directly in Excel. The commas will separate the data into columns, and each line of text will be a separate row. Most programming languages have functions to read CSV files into table-like data structures, such as Python Pandas DataFrames or MATLAB matrices.
4. NOTE: Remember the CSV file itself only stores data as plain text, so it cannot store things like formulas or plots. Once you have copied your data from the SD card, your first step will be to save the file in XLSX or another compatible format. This has the advantage of leaving an unaltered copy of the raw data. It is a good practice to keep a copy of this in case you make an error in your data analysis.

5. Let's modify our code from the previous activity to write some data in CSV format.
   5.1. First, change our filename String to DATA1.csv. We could leave the data as a TXT file, but then we would have to manually import the data into Excel or manually change the filenames. Using the CSV extensions will allow our computer to automatically open the data in Excel.
   5.2. Let's also add a variable to track which data line we are in the file.
   5.3. We do not need to change anything in our setup.
   5.4. In our loop, we will set up the data format. Usually, we want a number to identify the datapoint, and we can use our data_point variable for that. We will use the millis() function to timestamp the data. Let's use the reading from pin A0 for the first datapoint. We will take the value from A1 divided by 3 as a second data point. We do not need anything hooked up to A0 or A1, but let's assume that A0 is temperature and A1 is pressure.

   NOTE: A0 and A1 should still return nonzero values with no sensor connected; in a real situation, we would want to connect the sensor and perform a calibration using that analog pin. Then the calibration equation could be applied and the real value could be recorded in the data file.
   5.5. That means our format in the file will be **"data_point,time,temperature,pressure"**.
   5.6. We should create variables to store the temperature and pressure values. For temperature, since we are just using the raw ADC, we can use an integer. But since the pressure involves a decimal, use a float datatype.
   5.7. Now we need to modify our loop to read the two ADC values into the variable, open the file, write data, increase our data_point variable by 1, and close the file.

```
#include <SD.h>

#define CS_pin 10

File myFile; //Global file variable
String filename = "DATA1.csv"; //Name of our data file
int data_point = 0;  //Used to track what
                     //line of data we are in the file

void setup() {
  // put your setup code here, to run once:
```

Figure 1: Modified global variables to change the filename to a CSV and adding a variable to track the number of datapoints we have written.

```
void loop() {
  // put your main code h
  int temp;
  float press;
```

Figure 2: Sample variables for our "Temperature" and "Pressure" data.

```
if (myFile){
  temp=analogRead(A0);
  press=float(analogRead(A1))/3;
  //We need to convert the ADC value to a float
  //or the division will round the number
  Serial.println(filename+" opened");
  myFile.print(data_point);//Line counter
  myFile.print(","); //Comma for CSV
  myFile.print(millis());//Time in milliseconds
  myFile.print(","); //Comma for CSV
  myFile.print(temp);//Temperature from A0
  myFile.print(","); //Comma for CSV
  myFile.print(press);//Pressure from A1
  myFile.println();//creates a new line
  myFile.close();
  Serial.print("Wrote Datapoint ");
  Serial.println(data_point);
  data_point++; //Increase datapoint by 1
}
else{
  Serial.println("Failed to open file");
}
delay(1000); //delay slow the file down
```

Figure 3: Code for writing csv data to a file. Notice final serial.println() has been modified to tell use which datapoint has been written

NOTE: For this example, each of these steps is simple, but as your data becomes more complex, you will want to write separate functions for perform each of these steps.

5.8. Since we want this all to be in one line, when we print the data to the file, we need to use print() and the call println() at the end to create a new line. We also need to print the commas between each column of data.

	A	B	C	D
1	0	15	440	137.33
2	1	1032	391	128.33
3	2	2049	360	119.67
4	3	3065	379	124.33
5	4	4083	377	124
6	5	5100	365	120.67
7	6	6117	365	120.33
8	7	7134	365	120.33
9	8	8151	365	120.33
10	9	9167	364	120

```
1  0,15,440,137.33
2  1,1032,391,128.33
3  2,2049,360,119.67
4  3,3065,379,124.33
5  4,4083,377,124.00
6  5,5100,365,120.67
7  6,6117,365,120.33
8  7,7134,365,120.33
9  8,8151,365,120.33
10 9,9167,364,120.00
```

Figure 4: Sample data file when viewed in Excel(left) or in a text editor(right). Notice how Excel will drop any trailing 0s to the right of the decimal even though they are written in the data file.

5.9. Modify the code inside your if statement to accomplish these tasks. When complete, your code should look similar to Figure 3.

5.10. Upload your code and let it record several datapoints. Then look at the data file on your SD card.

5.11. You should have several data points like those in Figure 4. By default, the file should open in Excel if it is installed, but you can also view the data by right-clicking and opening it in a text editor, such as Notepad.

5.12. Look at the integer numbers for the "temperature" compared to the decimals for "pressure". Since the temperature was an integer, no decimal places were written. The pressure was always recorded to two decimal places. This can be controlled by giving the print() function the additional argument, as shown in Figure 5, where the number is the number of decimal places to be written. It is important to consider the required precision of your data and record a value that meets or exceeds that.

```
38    myFile.print(","); //Comma for CSV
39    myFile.print(press, 4);//Pressure from A1
40    myFile.println();//creates a new line
```

Figure 5: Modified pressure file write line, now the number will be recorded to 4 decimal places.

Working with Multiple Files

1. It is often useful and good practice to break your data up into multiple files rather than having a single giant file. This can break the data into smaller, more manageable pieces, separate data into different types, or provide some protection from data corruption.
2. In this example, we will break the data into multiple sequentially numbered files of 15 data points.
3. First, we need to break our filename string into multiple parts so we can easily change the name as we write data. We will have our name begin with "DATA", followed by a number

that we will count up, and then the extension ".csv". The filenames will be DATA1.csv, DATA2.csv, DATA3.csv, etc

4. We also need a variable to control the number of lines we write in each file. We need to create global variables for each of these, as shown in Figure 6.
5. Next, we need to write a function that will create a new file and change the filename variable so the next time data is written, it will be written in that file.
6. So the new_file() function needs to
   6.1. Increase the file_number counter
   6.2. Combine the three parts of the filename into the new complete filename
   6.3. Reset the data_point counter to 0 for the new file.
7. Write a function to do so. It should look similar to the code in Figure 7. Don't forget to include Serial.print() messages to inform you of what the Arduino is doing.
8. Now we just need to call the new_file() function in the correct places, and it will create the file and set that file to be the write location.
9. The first spot we need to call the function is at the end of the setup() so that the initial file is created.
10. Then, in our loop, we want to check if we have written the maximum number of data points in the current file. When this is true, we call the new_file() functions, which will create the new file and reset the data point counter. Add the code to do so at the end of your loop, as shown in Figure 9.
11. Finally, it would be good to add the file number to our data, that way we can tell the original file data

```
File myFile; //Global file variable
String filename; //Name of our data file
int data_point = 0;  //Used to track what
                     //line of data we are in the file

String filename_start= "DATA"; //Start of filename
String filename_end=".csv"; //End of filename
int file_number=0;
int max_lines =15; //Number of lines to
                   //write in a single file
```

Figure 6: The additional global variables to handle multiple files, we will use multiple strings to allow us to change the filename as we create new files. The number of data lines per file can easily be changed by modifying that variable.

```
65  void new_file(){
66    file_number=file_number+1; //Increases the filenumber by 1
67    filename=filename_start+String(file_number)+filename_end;
68    //Combines the parts of the file names together in the full string
69    //Since the writing part of the loop also uses this filename
70    //the data will be written to the new file
71    SD.open(filename);
72    Serial.print("Created file: ");
73    Serial.println(filename);
74    data_point=0; //Resets the datapoint counter
75  }
```

Figure 7: Function to create a new file.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Initializing SD card...");
  if (!(SD.begin(CS_pin))){   //Attempt to i
    Serial.println("Initialization Failed");
    while(1); //Stops the code here since
  }
  Serial.println("Initialization successful'
  new_file();

}
```

Figure 8: Calling newfile() at the end of setup().Failing to call the function here will result in trying to use an empty string for the filename, resulting in no data being written.

```
59    delay(1000); //delay slow the file
60    if (data_point>=max_lines){
61      new_file();
62    }
63  }
64
```

Figure 9: If statement in the loop to create a new file once the max number of datapoints is reached. Note the way this is setup, we will get points 0 to 14 in each file.

when multiple files are combined. This should also give each data point a unique ID if needed.

12. Simply add another myFile.print() function before the data_point print, like shown in Figure 10. Then do not forget to add a comma to separate them.

```
40    Serial.println(filename+" opened");
41    myFile.print(file_number);//File number counter
42    myFile.print(","); //Comma for CSV
43    myFile.print(data_point);//Line counter
44    myFile.print(","); //Comma for CSV
```

Figure 10: Adding the file number counter to the data format, now each datapoint is uniquely identified by a combination of file number and data number.

13. Now upload your code and watch the serial monitor as it records multiple data files. When you open the SD card, you should see multiple files, like in Figure 11.
14. If you open the files, you should see the first column matching the number in the filename and then the rest of the data, with 15 datapoints in each file.

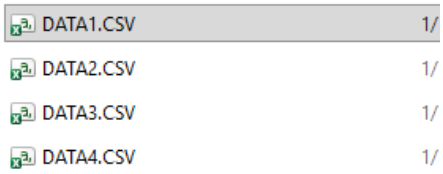

Figure 11: Multiple files written to the SD card

15. You may notice that some of the files have more than 15 datapoints; the data counter might repeat within a file.
16. Reinstall your SD card into the Arduino and hit the reset button while watching the serial monitor.
17. Let it record several data points and hit the reset button again. What do you see?
18. You should see that every time the Arduino resets, it starts writing data in the first file. If you look at that data file, you will see multiple data points of 0s, one for each reset.
19. A better solution would be to check the SD card for existing filenames and skip them if they already exist.
20. Let's modify the new_file() function to do so. We can use a while loop along with the SD.exists() function, as shown in Figure 12.
21. We can also write the data labels to the first line when we create the file. Notice that we need to actually open the file in write mode and close it to flush the data, just as we do in the loop, since we are now writing data, not just creating the file.

```
65  void new_file(){
66
67    String file_header("File Num,Datapoint,time,temp,press");
68    filename=filename_start+String(file_number)+filename_end;
69
70    while(SD.exists(filename)){
71      Serial.print(filename);
72      Serial.print(" Already exists trying ");
73      file_number=file_number+1; //Increases the filenumber by 1
74      filename=filename_start+String(file_number)+filename_end;
75      //Combines the parts of the file names together in the full string
76      //Since the writing part of the loop also uses this filename
77      //the data will be written to the new file
78      Serial.println(filename);
79    }
80
81
82    myFile=SD.open(filename, FILE_WRITE);
83    myFile.println(file_header);
84    myFile.close();
85    Serial.print("Created file: ");
86    Serial.println(filename);
87    data_point=0; //Resets the datapoint counter
88  }
```

Figure 12: Modified new_file() function. Now, each time it creates a new file, the function tries the numbers until it finds one that does not exist. This way, when the Arduino resets, it will go to the next file to write data. It also always writes the file_header as the first line so the CSV columns have labels.

22. Now upload your code (and clear the SD card). Allow the program to record several data files and hit reset. You should see something similar to Figure 13. If you look in the data files, you should see that the columns now have labels and data points are no longer repeated.
    Note: For this example, each data file is only about 15 seconds' worth of data. For a flight situation, you would most likely want to make the data files larger (30 minutes or 5 km worth of data).

```
DATA3.csv opened
Wrote Datapoint 2
Initializing SD card...
Initialization successful
DATA0.csv Already exists trying DATA1.csv
DATA1.csv Already exists trying DATA2.csv
DATA2.csv Already exists trying DATA3.csv
DATA3.csv Already exists trying DATA4.csv
Created file: DATA4.csv
DATA4.csv opened
Wrote Datapoint 0
DATA4.csv opened
Wrote Datapoint 1
DATA4.csv opened
```

Figure 13: Serial monitor output for the updated new_file() function. Notice how the last file before the reset was DATA3.csv and the Arduino resumed recording on DATA4.csv

You may also wish to include multiple header lines at the start of each file for documentation, for example you may include the date (since that should be the same for all data points) or the version number of the software that wrote the file.